Persistent Residual Increase in Server Processing Time

Mohamed Mansour[1] and Karsten Schwan[1]

The College of Computing at Georgia Tech, Atlanta GA 30332, USA
{mansour,schwan}@cc.gatech.edu

Abstract. In this case study we present our observations of a query processing engine running at a server farm operated by one of our industrial partners. We examine the query engine response time (termed MSBFS) under a variety of conditions. Observations show that there is a persistent residual increase in the server processing time that is only reset with rebooting the hardware.

1 Introduction

The configuration of each server is shown in Figure 1. We label each process as either persistent or non-persistent. A persistent process remains alive across several messages, while a non-persistent process is started to process a message, then terminated with a new process launched for the next message.

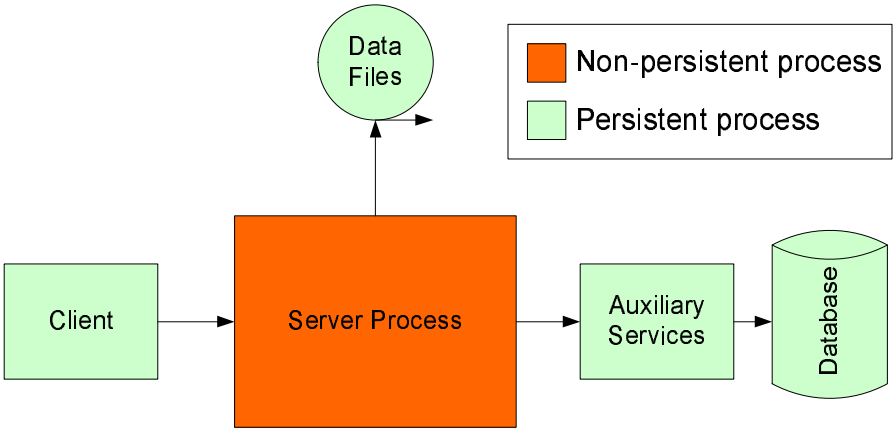

Fig. 1. Overview of Distributed System

The non-persistent setting is set by the system administrators and is the current setting used in the server farm. This choice reflects the complexity of the application code, since starting a new process for each request eliminates the

need to clean all related state in the process, a non-trivial programming task. Terminating and restarting a new process for each query has no impact on its actual processing time measured in the system. This is because (1) there are no server-resident request queues and (2) a server machine is considered unavailable and thus, no queries are sent to it during process termination, launch, and initialization. Query time is measured only once the process is fully initialized and ready to accept a new message, and a query is considered complete after a reply has been sent.

The inefficiencies implied by non-persistence are overcome by adding extra machines to the server farm and thereby maintaining the low overall farm utilization needed to provide predictable levels of service. The economies of this decision are deemed preferable to investing the additional programming and debugging time needed for correct operation with process persistence.

The complexities inherent in modern enterprise applications give rise to the fact that there typically remain unresolved programming issues with such applications. An example is the inability to operate in persistent mode for the server processes mentioned above. Facts like these create interesting technical problems to be addressed for large-scale enterprise applications, which are (1) to better understand the behavior of these applications under different operating conditions and (2) to find methods to control these behaviors to ensure predictable operation. In previous work [4], for instance, we presented a technique for understanding message sequences that trigger undesirable behaviors in certain servers, and we then created ways to prevent such server from affecting other enterprise services – performance isolation. The work shown here demonstrates two key facts. First, such undesirable behaviors exist even when applications are well-debugged, because of undesirable interactions between application and operation system, for example. We note here that the data needed to process queries are stored in map files. The server accesses the data in 64KB chunks using Microsofts Memory-Mapped files mechanism [1]. Second, since the root cause of these behaviors lies outside application boundaries, this limits the scope and applicability of existing reliability technique [2] and therefore, new techniques that focus on monitoring and control at the component boundaries [3] are needed to provide some degree of protection against such behaviors.

2 Experimental Evaluation

The following hardware is used in the experiments shown below. The machine used is an IBM eServer xSeries 225 running MS Windows Server 2003 Server Edition with SP1. The machine has dual Intel Xeon processors running at 3GHz each, with 3GB RAM and 136 GB SCSI disc with a NTFS filesystem. The server machine is isolated from the rest of the farm to prevent data updates during the experiment. Map files were obtained on 6/13/2006 and were used with server version R16 (revision 1.55.1.79.1.240). Map files were about 10GB in size. The server is coded in C/C++ and compiled with the Microsoft Visual Studio 6.0 compiler.

The experiment is conducted using a set of 14 messages. Each run consists of running the set 500 iterations. Each iteration uses a different randomized order.

2.1 Base Run

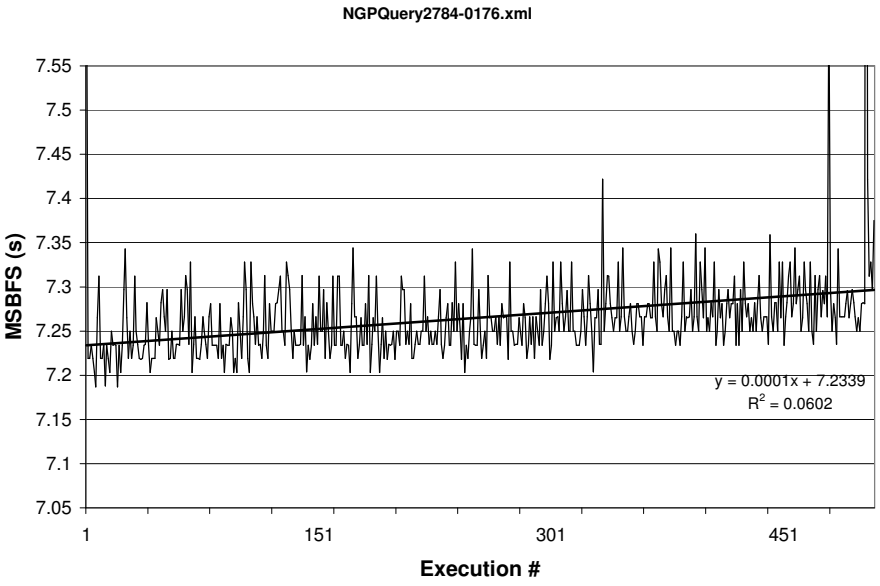

Fig. 2. MSBFS time for a sample query from Experiment 3 - base run

The set is executed about 500 times with the same randomization technique used in experiment I. 12 out of the 14 queries show a slight increase in MSBFS time and only 2 show a slight decrease in MSBFS time. We notice here that even though the MSBFS time is almost 7 times greater than the queries in experiment I, we still have a very slow gradual increase in the trend line. Figure 2 shows the plot of MSBFS over time for one sample query. The effect of this gradual increase on queries that return solutions after they time out needs to be investigated.

2.2 Recycle Auxiliary Services

The next step is to re-start all NGP services and repeat the experiment to ascertain whether or not MSBFS time resets. Results show that restarting the services does not affect the residual increase in MSBFS, thereby demonstrating the inability of techniques like micro-reboot to solve problems like these. Figure 3 plots the MSBFS time for the same sample query used above. Notice that the residual increase in MSBFS is not eliminated by restarting the services.

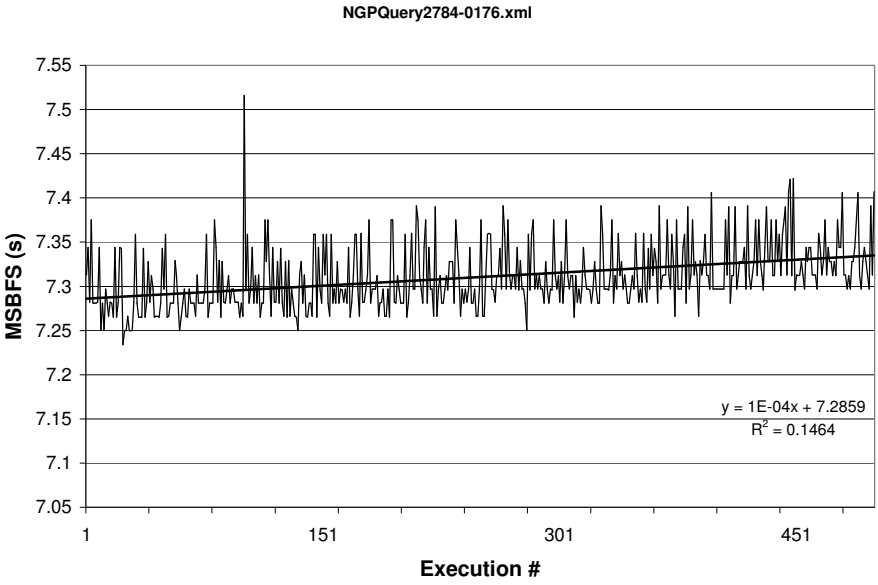

Fig. 3. MSBFS time for a sample query from Experiment 3 - recycle services

2.3 Use New Map Files

A potential factor causing residual increases is the update of Map Files. To investigate this, we rename the current map files directory, create a new one with the same name, and copy all files to the new directory. The effect of this copy operation is to trick the OS into thinking that this is a new set of map files.

Experimental results show that refreshing map files is not a factor in resetting the persistent residual increase in MSBFS time. Figure 4 plots the MSBFS time over 500 executions for a sample query, notice how the MSBFS time starts at the point where it left off in experiment 3 above.

2.4 Recycle SQLServer

To conclude experiments, we repeat the above experiment after recycling the SQL Server engine running on the test box. The service is stopped and then restarted using the SQL Server Service Manager. Figure 5 shows the MSBFS time for the NGPQuery2784-0176.xml. Note that the response time still carries the residual increase from the last experiment.

3 Conclusions and Future Work

We are still unable to find the exact cause for this persistent increase in processing time. The only action that seems to reset this residue is to reboot the

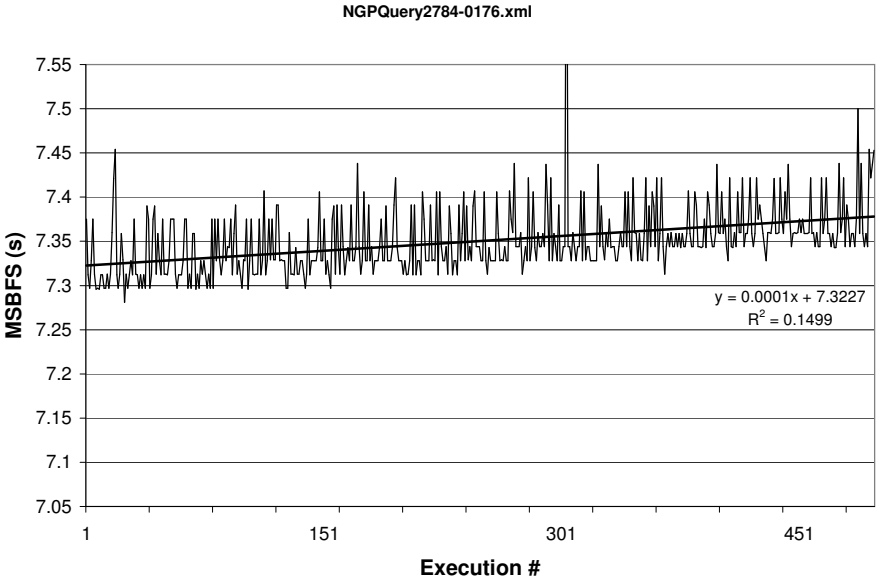

Fig. 4. MSBFS time for a sample query from Experiment 4 - refresh map files

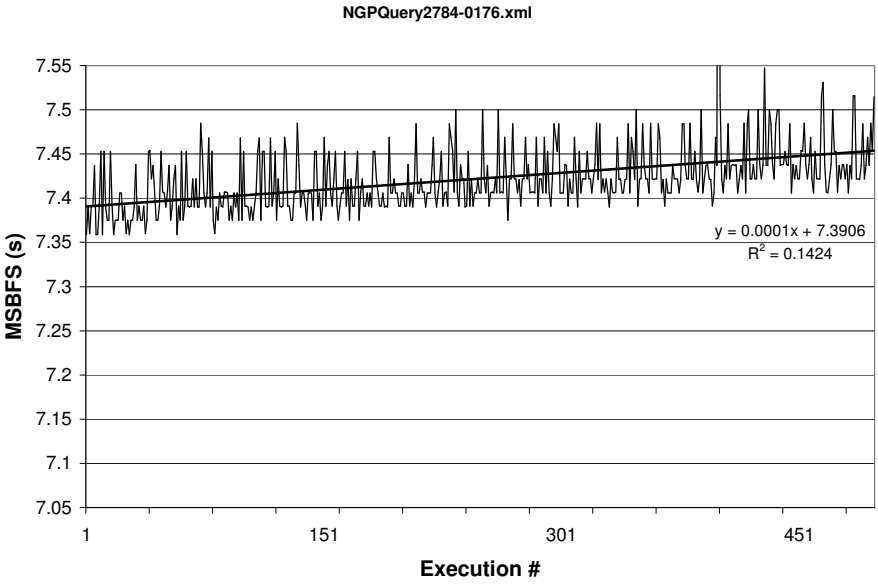

Fig. 5. MSBFS time for a sample query from Experiment 4 - recycle SQLServer

machine. We know for sure that it is not an application issue as the application is completely terminated and restarted for each query execution. We have experimented with auxiliary services (SQL Server and other daemons), and resetting them does not affect the residual increase. We also rule out that this is caused by our test driver, as the driver is always restarted between experiments. At this point we are led to believe that this is an Operating System issue and/or an issue caused by application/OS interactions.

The general insight derived from these experiments is that it is often difficult, if not impossible, to determine and then remove the root causes of undesirable behaviors in complex enterprise applications. It is an easier task, however, to determine the localities of such behaviors and then, to isolate them from other application components. This motivates our work on performance and behavior isolation in distributed enterprise applications.

References

[1] "Microsoft: Managing memory-mapped files in Win32." `http://msdn2.microsoft.com/en-us/library/ms810613.aspx`. [online; viewed:10/12/2006].

[2] Candea, G., Cutler, J., and Fox, A., "Improving availability with recursive microreboots: a soft-state system case study," *Perform. Eval.*, vol. 56, no. 1-4, pp. 213–248, 2004.

[3] Mansour, M. S. and Schwan, K., "I-RMI: Performance isolation in information flow applications," in *Proceedings ACM/IFIP/USENIX 6th International Middleware Conference (Middleware 2005)* (Alonso, G., ed.), vol. 3790 of *Lecture Notes in Computer Science*, (Grenoble, France), Springer, 2005.

[4] Mansour, M. S., Scwhan, K., and Abdelaziz, S., "I-Queue: Smart queues for service management," in *Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC 06)*, Lecture Notes in Computer Science, (Chicago, USA), Springer, 2006.